PLATFORM9

KubeVirt: How to Run VMs on Kubernetes

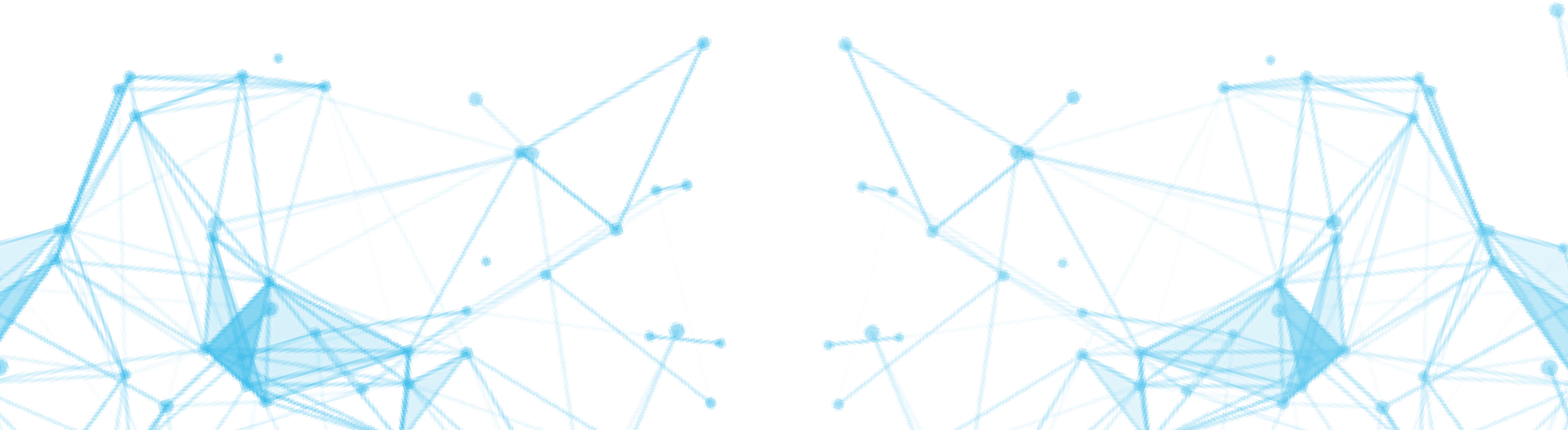


Table of Contents

In the world of DevOps and software development, one of the challenges is bridging the gap between your containerized environments and your virtual machine environment.

While it's not impossible, configuring the connection between your containers and your other services isn't trivial either. Fortunately, you can use KubeVirt to bring your virtual machines into your container environment, which opens a whole new world of possibility and ease. In this guide, we're going to introduce you to a new solution from Platform9 that allows you to deploy virtual machines (VMs) on your Kubernetes cluster. With VMs, you can now deploy non-containerized applications and run them alongside your cloud-native applications, on a single Kubernetes cluster.

What is KubeVirt?

KubeVirt is an open-source project that allows you to create virtual machines within a Kubernetes cluster. The project was initiated by RedHat in 2016. It has since been open-sourced and was inducted into the Cloud Native Computing Foundation's (CNCF) Sandbox Project, on September 9th 2019. KubeVirt is a collection of custom resource definitions (CRDs) and controllers. With KubeVirt, you can use the Kubernetes platform to run and manage application containers and VMs side-by-side. Under the surface, KubeVirt leverages QEMU and KVM to power the virtual machines. An overwhelming advantage of this approach is that all the tools and utilities you use to manage and maintain your cluster, apply to the KubeVirt VMs as well. You can use the native Kubernetes functions to set up and control:

- Scheduling
- Storage
- Networking
- Monitoring
- Orchestration

Let's start by looking at some examples of challenges you can solve with KubeVirt. Once we've done that, we'll look at KubeVirt itself in more detail and see how it fits into the Kubernetes ecosystem. Finally, we'll walk through the steps to get up and running with a KubeVirt deployment of your own on your Kubernetes cluster.

Why Use KubeVirt?

The concept of being able to run VMs inside of your Kubernetes platform is intriguing, but the power comes from the different business problems it can solve. Let's look at some of the business problems KubeVirt solves or at least simplifies.

Unified Orchestration Platform

Running both your VMs and containerized applications within a Kubernetes cluster simplifies your orchestration needs. Instead of configuring a CI/CD pipeline for your containers and one for VMs, you can combine them and manage them all from a single tool, such as Helm. The configuration of administrative tooling, networking, and monitoring are also brought in under the umbrella for both application types, dramatically simplifying your systems.

Application Modernization

Anyone who has attempted an application migration from a monolithic architecture to micro-services knows that it can be especially challenging. A big part of that challenge is facilitating interoperability between the legacy application and new micro-services as they are created and deployed. Deploying all of the players – both legacy and new – to the same platform using both VMs and containers, significantly reduces the number of hurdles.

Virtual Network Function Modernization

Organizations that handle intensive I/O workloads might be using Network Functions Virtualization (NFV) to optimize their systems for that work. These systems make use of custom kernel modules, network drivers, and other characteristics that prevent their migration to a containerized architecture.

Moving your NFV workloads into VMs with KubeVirt allows you to move to Kubernetes, and host your NFV VMs alongside other already containerized applications.

Kubernetes on Bare Metal Kubernetes

A recent trend in the quest for more performant Kubernetes has been to deploy Kubernetes on a bare-metal server. If you're not familiar with the concept of a bare-metal server, it's a deployment directly on a host machine, without the constraints of a virtual machine. It's a bit like the original concept of a server, before the advent of virtualization. This approach allows such advantages as taking advantage of hardware accelerators and GPU processing power.

KubeVirt Compute Concepts

Now that we have explored potential use cases for KubeVirt, let's take a deep dive into more of the details and concepts in use with KubeVirt. We'll start by exploring some of the concepts related to compute functionality within KubeVirt.

Virtual Machine (VM)

When we talk about a VM with KubeVirt, we're referring to a custom Kubernetes object. The CRD for the object contains the specification for the Virtual Machine Instance, which we'll talk about next. The VM serves as an abstraction layer about your VM instances that can communicate the status of running/not running and other labels. The VM object is not associated with a specific pod or process.

Virtual Machine Instance (VMI)

The VMI is also a custom object. It is an object that represents a single running virtual machine instance and consists of two parts. The first part of it holds information to make scheduling decisions. The second part holds information about the virtual machine application binary interface.

VMI Preset

The VMI Preset is similar to the idea of flavors or instance types in other VM technologies. In AWS, the EC2 instance type prescribes the configuration for VMs of that type. The VMI Preset contains specific configurations for:

- Memory
- CPU
- Storage
- Networking

True to its nature as a Kubernetes component, you assign a label to the VMI Preset, and if the label on the VMI matches, then the VMI inherits the configuration of the VMI Preset. Where it differs from the traditional idea of a VM Instance Type or flavor, is that the VMI can override any of those settings. When Kubernetes creates a new VMI, the related VMI Preset provides the default settings for the instance. Any settings that have an override value are then updated to that value before fully provisioning the VMI.

KubeVirt Boot Options

With our understanding of the nature of the VM and VMIs with the Kubernetes ecosystem, let's look at the different options you can choose to define the state of your VM as it boots up.

Ephemeral Disk Boot Option

Booting up a VMI with an ephemeral disk is useful in any scenario where disk persistence is not desired or essential. KubeVirt dynamically generates the ephemeral images associated with a VM when the VM starts. The storage shares the lifecycle with the VM, meaning that it is lost when the VM is terminated or restarted.

Persistent Disk Boot Option

If the persistence of state after the termination or rebooting of your VM is essential for your operations, then the Persistent Disk option is the option to select. In this case, the VM attaches a persistent data volume.

KubeVirt Containerized Data Importer (CDI)

The question you might now be asking is:

> **"How do I load compatible images for my KubeVirt VMs?"**

Allow us to introduce you to KubeVirt's Containerized Data Importer (CDI). KubeVirt CDI is a utility designed to import, upload, and clone Virtual Machine images for use with KubeVirt.

KubeVirt CDI is a CRD that sits on top of the Persistent Volume Claim (PVC) and enables the process of importing an external virtual machine image, and then storing it within the cluster storage. This stored image then becomes a *Golden Image*, which KubeVirt can reference when provisioning a new VM, or additional VMIs, based on that image. CDI can accept QCOW2 and RAW image formats.

If you would like to learn more about KubeVirt's CDI utility, you can go to **Containerized Data Importer**. KubeVirt CDI is a second open-source project under the KubeVirt banner, which can be accessed and downloaded from the **Containerized Data Importer GitHub repository.** You can also reference the image below, which illustrates a typical use case for CDI with a Kubernetes cluster.

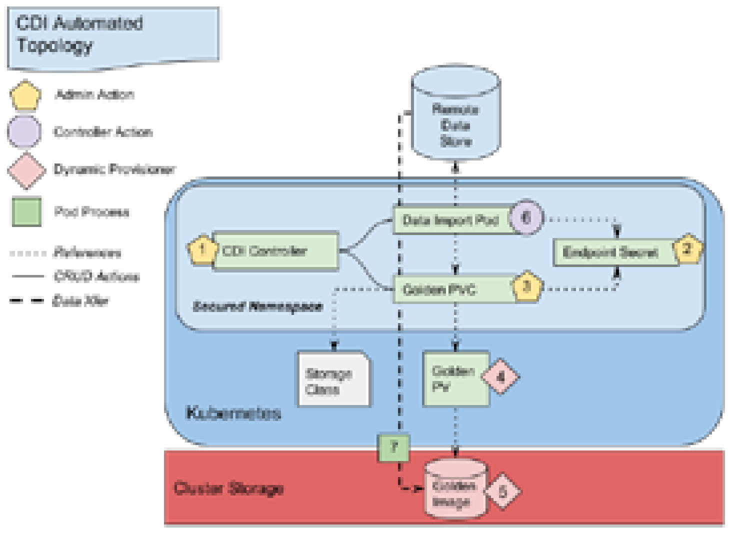


Fig. 1 Common Use Case for CDI to Import and Use a New Image (Source: **Beyond Containers**)

KubeVirt Storage

KubeVirt supports the following storage options:

CloudInit NoCloud and ConfigDrive

KubeVirt supports both CloudInit NoCloud and CloudInit Config Drive. CloudInit NoCloud and ConfigDrive are provisioned within the startup scripts of the VMI. The script defines a disk and a volume for the NoCloud and ConfigDrive data source. CloudInit NoCloud helps to set up storage devices to configure SSH access keys and many other aspects of a system. (**Creating Virtual Machines I cloud-init**)

emptyDisks

An emptyDisk is an initially empty disk, as the name implies. You mount the emptyDisk on the same or different path in each container. This approach is similar to an EmptyDir in Kubernetes. An emptyDisk exists for the lifetime of the Kubernetes pod in which it resides.

When an emptyDisk is attached to a virtual machine, an extra sparse qcow2 disk is allocated and lives as long as the virtual machine. It can survive guest side virtual re-boots but not virtual machines re-creation.

hostDisks

The hostDisk option has two usage types, DiskOrCreate and Disk. By using hostDisks, you can create or use a disk image located somewhere on a node. The difference between the two types is that DiskOrCreate creates an image if a disk image does not exist at a given location, while Disk type states that a disk image must exist at a given location.

containerDisks

Formerly known as registryDisk, it is an ephemeral storage device that can store and distribute VM disks in the container image registry. You can assign the disk to any number of active VirtualMachineInstances in the VM's disk section of the VirtualMachinesInstance spec. containerDisks are great for users that want to replicate a large number of VM workloads that don't require persistent data, while not so great for any workload that requires persistent root disks across VM restarts. (**containerDisk**)

dataVolume

KubeVirt also supports dataVolume, which is a method to automate importing virtual machine disks onto PVCs. DataVolumes can be defined in the Virtual Machine specification, directly adding the DataVolumes to the dataVolumeTemplates list. (**Disk and Volumes**)

Kubernetes Primitives

Finally, KubeVirt supports Kubernetes Primitives, as well as Live Migration. Kubernetes building blocks such as ConfigMap, Secret, and ServiceAccount are all available in KubeVirt.

One thing to note is that these storage changes generally require a restart of the Virtual Machine. Restarting the VM allows it to recreate the VirtualMachineInstance object and, after that, recognize the attached store options.

Live Migration

Live Migration is the process wherein a running Virtual Machines Instance moves to another compute node while the guest workload continues to run and remains accessible. You can successfully do a live migration in KubeVirt when you have configured the storage and networking correctly.

Live Migration must be first enabled in the feature-gates to be supported. The feature-gates field resides in the kubevirt-config config map, and you can add Live Migration to it after you expand that field. (**Installation & Administration | Live Migration**)

Some situations you may encounter with Live Migration are:

- VMs using a PVC must have a shared ReadWriteMany access mode to be Live Migrated.
- VMs not using persistent storage, such as *containerDisks* may be Live Migrated.
- Live Migration is not allowed when the VM's pod network uses a bridge interface (*See Fig. 2 below*). Note: The default network interface type is a bridge interface.

Other interfaces, such as those granted by Multus (*see KubeVirt - Networking below*) may use a bridge interface for the purposes of live migration.

```
spec:
  networks:
   - name: network1
     pod: {} # pod network
```

KubeVirt Networking

By default, the VMs you create with KubeVirt use the native networking already configured in the pod. Typically, this means that the bridge option is selected, and your VM has the IP address of the pod. This approach makes interoperability possible. The VM can integrate with different cases like sidecar containers and pod masquerading. When using pod masquerading, there is a defined CIDR chosen by yourself for which VM's are not assigned a private IP, and instead use NAT behind the pod IP.

Multus is a secondary network that uses **Multus-CNI**. Multus allows a user to attach multiple network interfaces to pods in Kubernetes. If you use Multus as your network, you need to ensure that you have installed Multus across your cluster and that you have created a NetworkAttachmentDefinition CRD. (**Creating Virtual Machines I Interfaces and Networks**)

While Multus has become the de-facto standard, another option you have is Genie. Genie is similar to Multus in that it assumes you have installed Genie across your cluster and that it is a secondary network. Genie uses **CNI-Genie**, which enables Kubernetes to connect to the choice CNI plugins installed on a host, including CNI plugins like SR-IOV. At the time of writing, the CNI-Genie project hasn't had updates made to it in over a year.

KubeVirt Architecture

When you use KubeVirt to create a new VM, the first step that happens is the creation of a new pod. Each VM resides inside its pod, and within that pod you have at least a *Volume Container* and a *Computer Container*.

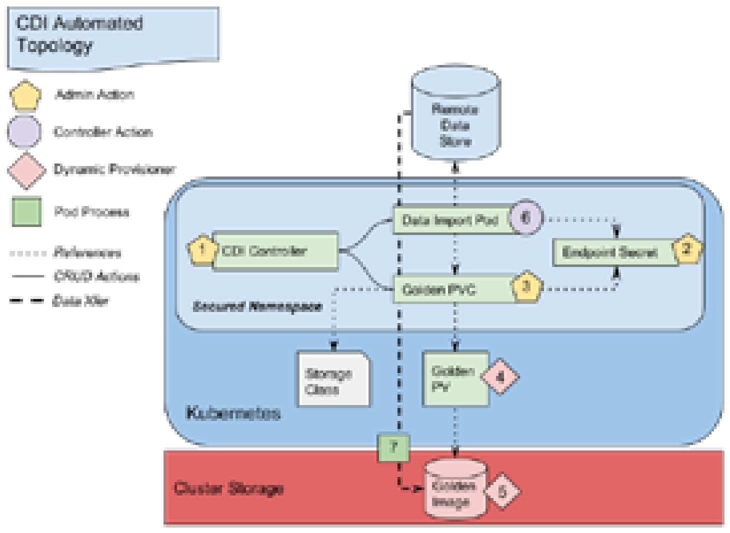


Fig. 3 Common Use Case for CDI to Import and Use a New Image (Source: **Beyond Containers**)

Volume Container

The *Volume Container* examines the embedded *Virtual Machine* Image inside the docker image. Once it has identified the image, it extracts it and then passes it on to the compute container. Essentially, the *Volume Container* is responsible for locating and passing the virtual machine disk image to the *QEMU* process and other Containers such as Sidecars, and Infra containers.

Compute Container

The *Compute Container* is what runs the actual VM. Within the Compute Container, there is the *Virt-Launcher*, Libvirt and finally the VM itself. The *Virt-launcher* is responsible for interacting with *Libvirt*. It launches *Libvirt* as a child process, and it is what *Virt-Launcher* uses to manage the life of the virtual machine.

Libvirtd

Libvirtd is the tool that runs the virtual machine using QEMU and KVM, where it is available. It is the standard component used to spawn virtual machines on Linux and provide the features that the user has requested. These features include memory limits, CPU limits, and specifications for any devices connected to the virtual machine.

Libvirtd can also receive requests from other components to stop and start the VM. Unlike containers that start by default, VMs can be dormant and may remain in that state until they start. Libvirtd provides this capability.

The continued development of KubeVirt will enable more features in the future. Some of these features may include support to hot plug or unplug devices, memory, and disk.

Networking and the VMI

KubeVirt assigns the Pod IP address to the VM running inside the Virt-Launcher POD by default. When the system provisions a new pod, the default networking setup begins with the creation of a veth pair. The veth pair connects Container Network Interface (CNI) to the Compute Container, which contains the VMI. If you log into the container and then query for the ethernet devices like using "ip a", you'll see this as eth0. The virt-launcher then creates a bridge, creates a tap device, and takes the IP address that is assigned to the container and then assigns it to the virtual machine that is running inside the container.

This specific approach results in the IP address of the pod being given to the enclosed VM. Once this setup is complete, the VM functions just like any other container within the cluster. The cluster treats the VM as just another ***workload***.

As stated above, this is the default setup, and you can replace it with a different networking strategy, such as NAT or masquerading.

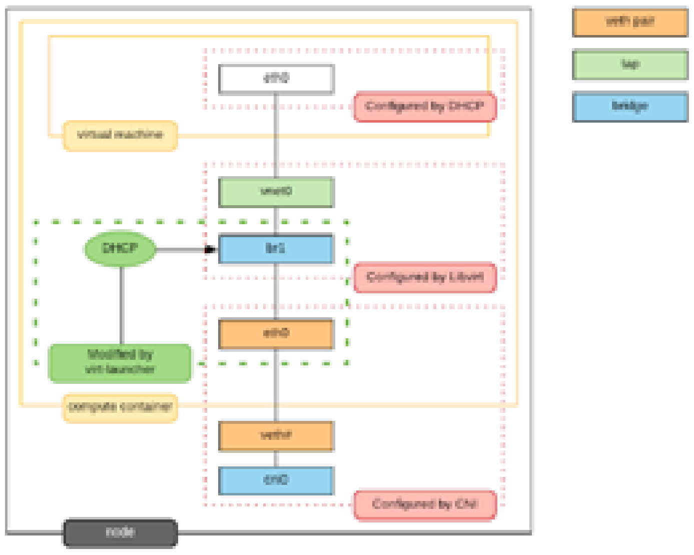


Fig. 4 Default Network Configuration Between CNI and the VM (Source: KubeVirt Networking Diagram)

Exploring Some of the Controllers

Each of the objects involved in supporting the VM has an associated controller. Given that KubeVirt as a project is still in its infancy, it is reasonable to expect additional objects and functionality to become available over time. The idea and concept of VM Group is an example of one of the ideas currently being discussed.

Here is a quick overview of some of the objects and controllers involved in supporting a KubeVirt VM. Currently, you can find Controllers for each of the following objects:

- VirtualMachine
- VirtualMachineInstance
- VirtualMachineInstanceReplicaSet
- Node
- Migration

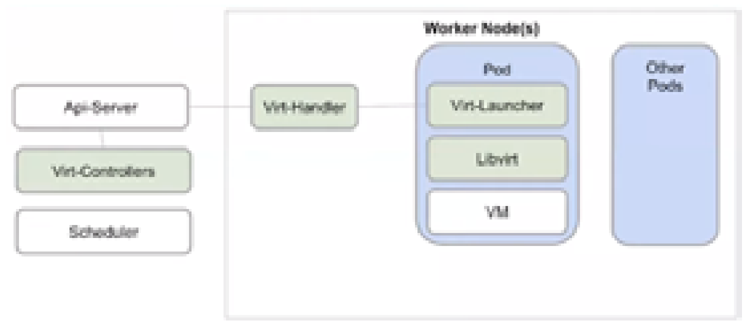
The VirtualMachine controller delegates most of the work it does to the VirtualMachineInstance controller, and reports on the state of the machine, among other things. The VirtualMachineReplicaSet controller ensures that workloads or VMs are always up and ready to go, and the Migration controller handles the process of migrating the VM to another node, if the need arises.

The Virt-Handler Daemonset

Finally, let's look at the Virt-Handler. The Virt-Handler is a Daemonset, which operates within every KubeVirt node in the cluster. This Daemonset is responsible for communicating with the Virt-Launcher through the UNIX domain sockets.

The Virt-Launcher, Virt-Controller and Virt-Handler Daemonset work together as the core system behind KubeVirt.

Fig. 5 A Holistic View of a KubeVirt System (Source: Beyond Containers)



Putting it into Practice

Now that we have a good understanding of the KubeVirt ecosystem, and how it fits into a broader view of the Kubernetes Cluster, you're well-equipped to begin experimenting with a KubeVirt deployment of your own.

> **One important thing to note before you begin your KubeVirt experiments: Since KubeVirt uses Linux QEMU and KVM by default, you'll need to run it on bare-metal servers.**

If that is not an option, you can also run it in a nested virtualization mode by using binary translation. This particular configuration is slower but sufficient for a proof of concept or demo purposes.

An excellent place to start is the KubeVirt project itself. The website includes detailed installation instructions, system requirements, and configuration information. The appendix of this white paper includes the steps required to set up KubeVirt on the Platform9 free tier. Please ensure that you install PMKFT with Flannel networking.

Learning More

Finally, keep an eye on the Platform9 blog for upcoming posts on KubeVirt; and general information, best practices, and performance tips for your Kubernetes clusters. You can even subscribe and have the latest posts delivered to your inbox.

KubeVirt is still in its infancy, and as it matures, the power of being able to run Virtual Machines alongside containers solves many problems that DevOps engineers currently face.

Appendix

See **GitHub** for more details.
Special thanks to Platform9 SE Clement Liaw for his technical contributions to this paper

PLATFORM9

Freedom in Cloud Computing

Platform9.com/contact

Headquarters
2465 Latham Street
Suite 110
Mountain View, CA 94040
650-898-7369
info@platform9.com